



Office de la propriété  
intellectuelle  
du Canada

Un organisme  
d'Industrie Canada

Canadian  
Intellectual Property  
Office

An Agency of  
Industry Canada

*Bureau canadien  
des brevets  
Certification*

La présente atteste que les documents  
ci-joints, dont la liste figure ci-dessous,  
sont des copies authentiques des docu-  
ments déposés au Bureau des brevets.

*Canadian Patent  
Office  
Certification*

This is to certify that the documents  
attached hereto and identified below are  
true copies of the documents on file in  
the Patent Office.

Specification and Drawings, as originally filed, with Application for Patent Serial  
No: **2,414,632**, on December 18, 2002, by **LOGICVISION, INC.**, assignee of  
Benoit Nadeau-Dostie and Jean-François Côté, for "Method and Circuit for Collecting  
Memory Failure Information".

*Gracy Pouches*  
Agent certificateur/Certifying Officer

October 20, 2003

Date

Canada

(CIPO 68)  
04-09-02

OPIC  CIPO

# Method and Circuit for Collecting Memory Failure Information

**Inventors** Benoit Nadeau-Dostie, Jean-Francois Cote

**Abstract** A method of collecting memory failure information in real time for memories tested using an embedded memory test controller for the purpose of process monitoring, yield enhancement, redundancy analysis and bitmap generation.

<b>C n t e n t s</b>	
n	<b>Background of the invention . . . . . -2</b>
n	<b>Summary of the invention . . . . . -5</b>
n	<b>Brief description of the drawings . . . . . -6</b>
n	<b>Detailed description of the preferred embodiments . . . . . -7</b>

## Background of the invention

# Background of the invention

## 1. Technical field

This invention relates to a method of collecting failure information in real time for memories tested using an embedded memory test controller for the purpose of process monitoring, yield enhancement, redundancy analysis and bitmap.

## 2. Definition of the problem and objectives

Memory failure information can be used to monitor and improve the quality of an integrated circuit manufacturing process. This is done by diagnosing and correlating functional failures to defects in the circuit introduced during manufacturing. The defects can in turn be associated to a certain step of the manufacturing process. Modifications can be made to this step by using different settings for temperature, duration, dust control, etc... If these modifications are not sufficient or possible, the design itself might have to be changed to be more tolerant of this type of defect. One type of design modification is the introduction of redundant memory cells that can be substituted to defective ones.

Conventional diagnosis methods require to send detailed information about all failures to the tester where it will be analyzed. It is difficult or even impossible to do so because of the bandwidth required. There are 2 aspects to the bandwidth problem. The first aspect is related to the amount of information to be sent. One bit of information is required for every memory bit read. Memories often have a large number of bits (16 to 256) that are read simultaneously. This number of bits is multiplied by the number of words and by the number of read operations. This large amount of detailed information can be sent off-chip via a large number of pins corresponding to the number of bits in a word. However, it is not desirable or even possible to do so. It is well known that the number of pins must be minimized to reduce test cost.

The second aspect of the bandwidth problem is the rate at which the detailed information can be sent. Memories are now operating at clock rates that exceed tester interface clock rates. This is why embedded test controllers are used to perform the test. Embedded test controllers can determine whether the memories are good or bad. However, raw detailed failure information can not be transferred without transformation because it is generated faster than what the tester can accept. One possible solution is to use a demultiplexer that will reduce the rate of transfer to the tester. However, it requires to multiply the number of pins used to transfer the failure information by the ratio of the memory clock rate on the tester interface clock rate. This ratio can be as high as 10.

Clearly, a different method is needed to compress the failure information that needs to be transferred without sacrificing the ability to extract the relevant failure information. The level of resolution of the information can be traded off based on the application. For example, it is sufficient to know the density of failures for yield analysis (i.e. number of failures in a column or row for example) for any density of failures whereas it is necessary to know more precisely the location of individual failures when the density is low for repair

## Background of the invention

analysis. The method supports such trade-off. The method takes advantage of the memory structure and certain characteristics of conventional memory tests to generate failure summaries.

Figure 1 shows a memory containing defects and where black squares are used to indicate bits that appear to be defective after applying a test. This memory contains 128 words of 4 bits each for a total of 512 bits. These bits are organized in arrays. Each bit-array has a number of rows and columns. Each bit of an array is accessed by applying a row and a column address. In our example, 4 bits are used to access one of the 16 rows and 3 bits are used to access one of the 8 columns of each bit-array. There are 4 groups of 2 columns because words have 2 bits each. The bank address bit is used to select which array or bank is accessed. Actual memories usually have a lot more rows, columns and banks.

In the first bank (bank 0), all bits of column 2 (bit 1) are marked. In the second bank (bank 1), all bits of row 4 are marked. However, the actual defects are in the circuitry used to access the bits. For example, a single defect in a bit line could be responsible for the first set of failures. A single defect in a word line could be responsible for the second set of failures. Being able to classify these failures on-chip allows to compress the amount of information to be transferred to the tester considerably.

Figure 2 shows a representative set of failure patterns that are of interest from a process monitoring point of view because each pattern can be associated with the presence of specific defects. They generally consist of single cell failures, 2-cell failures, partial/full/two column failures and row failures. It is not necessary to be able to classify all these patterns on-chip and different classifications are possible. However, the objective of the method is to provide an embedded test infrastructure to transfer compressed failure information. The method allows a trade-off between the number of failure patterns that can be classified according to a classification like the one shown in figure 2 and, the amount of on-chip circuitry required to do so and the additional amount of test time that is required to perform the classification.

### 3. Description of the related art.

Chen et al, 'Enabling embedded memory diagnosis via test response compression', pp. VTS'01: (see also patent application WO 01/67463 A1): compression technique uses a 6-bit output per group of fail vector. Bits of the fail vectors are combined in various ways along rows (AND, OR, 2OR (2 or more failures in that word)), columns (MaskedAND, MaskedOR, Repeat) and diagonals (XOR). The main issues are that high-speed outputs are needed and the complexity of the functions require to break the fail vector in many groups, increasing the number of pins that need to be connected to the tester.

Schanstra et al, 'Semiconductor manufacturing process monitoring using BIST for embedded memories', pp. ITC'98: This method uses several registers to collect failure information during the execution of the memory test. The registers are only inspected at the end of the memory test. The registers include a fault

### Background of the invention

counter, a column fault capture unit, an address capture unit for isolated faults. The issues are that it doesn't capture information about faulty row and the results of the column fault capture unit are corrupted in the presence of faulty rows. It is restricted to algorithms that uses column access mode only. It requires too many registers because the test controller needs to accumulate the failure information until the end of the test instead of sending failure information as it is available.

## Summary of the invention

## Summary of the invention

This document describes a method of collecting memory failure information in real time for memories tested using an embedded memory test controller for the purpose of process monitoring, yield enhancement, redundancy analysis and bitmap generation.

The method generally consists of generating a failure summary for each column during memory test phases using a column access mode and for each row during memory test phases using a row access mode. Memory test phases are executed at a first clock rate while failure summaries are transferred from the memory test controller to an external tester at a second, lower, clock rate. Transfers are synchronized at the beginning of each column or row. Transfers are performed concurrently with the memory test. Failures are categorized into types and a count of each type of failures is kept. Failure address registers store the row or column address of selected failures. A test mask register indicates which memory data outputs failed a comparison with an expected data value during the execution of a test on a column or row. The test mask register is initialized at the beginning of each column, in column access mode, or row, in row access mode. Failure summaries include a combination of address information, failure count information and failure mask register information. Each information is encoded to minimize the amount of information to transfer.

Brief description of the drawings

## **Brief description of the drawings**

Figure 1 “Memory configurations and failure pattern examples”

Figure 2 “Bitmaps of interest and potential failure causes (SRAM/DRAM)”

Figure 3 “Memory test controller architecture”

Figure 4 “Failure summary generator architecture”

Figure 5 “Failure summary generator flow diagram (column and row access modes)”

Figure 6 “Timing of failure summary transfer”

Figure 7 “Failure summary: Example 1”

Figure 8 “Results: Example 1: Complex bitmap interpretation (bit grouping = 1)”

Figure 9 “Failure summary: Example 2 (no row access mode)”

Figure 10 “Fail mask and fail mask register”

Figure 11 “Detection of single-bit and multiple-bit failures (first stage)”

Figure 12 “Detection of single-bit and multiple-bit failures (second stage)”

Figure 13 “Cascading single/multiple-bit detectors”

Figure 14 “Failure summary: Example 3”

## Detailed description of the preferred embodiments

**Detailed description of the preferred embodiments**

Figure 3 shows an example of a memory test controller connected to a memory. The memory test controller is typically embedded in an integrated circuit. The memory can be embedded in the same integrated circuit or could reside off-chip. The functional connections to the memory are not shown to simplify the figure. The memory test controller can be shared among several memories.

The memory test controller is composed of several blocks. All blocks are controlled by a first clock (Clock). The general control block interacts with all other blocks as well as with an external tester either directly or through a test access port. In general, the interaction with the tester is limited to initiating a memory test and, optionally, collecting failure information. Both operations require setting registers (group of memory elements) of the test controller to appropriate values and reading registers containing relevant information after the execution of the memory test.

The general control block determines the sequence of read and write operations that are to be performed to test the memory. The interaction of the general control block with the R/W control, address generator and data generator as well as the connections between the memory test controller and the memory are well known in the art and are not discussed here. We will limit the discussion to modifications to the comparators block and the failure summary generator block and its interaction with the other blocks as these are the only new elements required to implement this invention.

The sequence of read and write operations is determined by memory test algorithms which are well known in the art. These algorithms are divided into phases. Several of these algorithms are such that, during a phase, all memory locations are accessed in a column access mode or row access mode. In column access mode, a same sequence of read and write operations is applied to all memory locations of a column, one location at a time, before another column is accessed and the sequence is repeated until all locations have been accessed. Similarly, in row access mode, a same sequence of read and write operations is applied to all memory locations of a row, one location at a time, before another row is accessed and the sequence is repeated until all locations have been accessed.

The method described in this document assumes that at least some phases of the memory test algorithm use a column or row access mode. During these phases, failure summary data is generated for each column (or row) on-chip and transferred off-chip for further analysis. The failure summary generator receives various inputs from the comparators block at the system clock rate but transfers failure summary at the tester interface clock rate which is usually significantly lower.

The failure summary generator receives a synchronization signal (SyncPulse),

The method can also be adapted to more complex phases of an algorithm that access each location multiple times during the same phase. An example of this would be

### Detailed description of the preferred embodiments

an algorithm called "bit-surround" where, for each reference location, the reference location itself as well as all locations surrounding it are accessed. Since all locations are used as reference, each location is accessed more than once during a single phase of the algorithm. It might be preferable to only consider the failure information related to the reference cell to simplify the generation of the failure summary during this phase.

The on-chip classification of some of the double failures (2-cell, 2-column or 2-rows) shown in figure 2 might require a mode of the test controller that performs address mapping (or scrambling) so that 2 consecutive accessed cells, columns or rows are physically adjacent in memory. If this mode is not available or enabled, the identification of these failure will require additional circuitry or will need to be performed by a computer off-chip.

Also, for memories with multiple blocks, it is preferable to generate summaries for individual blocks. Multiple serial outputs can be used to transfer failure summaries corresponding to different blocks at the expense of extra registers in the failure summary block.

### Failure summary generation

The general algorithm used to generate summary data is described in "Failure summary generator flow diagram (column and row access modes)". It is illustrated for phases of an algorithm using a column access mode but the algorithm is the same for phases using a row access mode. At the beginning of each test phase, optional phase summary registers are initialized. At the beginning of each column, column summary registers and the fail mask register contained in the comparators block are initialized. The fail mask register has a memory element corresponding to each comparator and indicates whether a failure occurred at that comparator since the last time it was reset. A reset will typically occur at the beginning of each test phase but could also occur a few times during a test phase if summaries are generated for portions of the column (or row).

Then read and write operations are performed. During read operations, the memory data output is compared to an expected value provided by the general control block. If the data is the same, the next memory location in the column is accessed. If the data is different, a failure is detected and processed. We first determine if the failure is massive i.e. a large number of data output bits fail. There is a number of ways to define a massive failure and it will be discussed in more detail later. Massive failures are processed differently than other non-massive failures. The main difference is that the fail mask register is not updated for massive failures. This is because the failure is most probably due to a defect in the access mechanism to the entire location and not due to a defect of individual bits. The fail mask register information is only updated when only a few bits of the word are failing.

After processing of the failure, massive or non-massive, appropriate failure

## Detailed description of the preferred embodiments

count(s) are incremented. The failure counts considered at this point are: total number of failed locations, total number of failed locations with massive failure, total number of failed locations with non-massive failures. Within the last category, we can also have a separate count for single-bit vs multi-bit failures. A special circuit is required to discriminate these two cases and is described later.

After incrementing the failure count(s), another location is accessed unless all locations have been accessed. The processing of the failures might be sufficiently complex to require more than one clock cycle and can take place in parallel with the access of the next location so that there is no interruption of the test.

### Timing of failure summary transfer

Figure 5 shows the detailed timing of the transfer of the failure summary to the tester. The first waveform (Clock) shows a first clock used to perform the memory test. Ideally, the clock period of this first clock is substantially the same as the clock period used during the normal mode of operation of the memory. The second waveform (ExtClock) shows a second clock used to synchronize the transfer of the failure summary to the tester. The period of the second clock is longer than the one of first clock. An asynchronous interface similar to the one described in US patent 5,900,753 takes the first and second clock as input and generates the synchronization pulses SyncPulse. These pulses indicate when the serial input can be sampled using the first clock. The serial input is sampled until a start bit is detected. The start bit is set by the tester at the end of each column and/or row. Once the start bit is detected, the failure summary data (described later) is copied in a shadow register and shifted out under the control of pulses of the Shift/Hold signal generated by a small FSM (not shown) in the failure summary block. The Shift/Hold signal is in fact a gated version of the SyncPulse signal. A bit of the failure summary is output on each pulse of Shift/Hold until all bits of the summary have been shifted out.

In this simple example, the failure summary only has 4 bits but there could be more or less. The maximum number of bits of a failure summary is determined by the time it takes to test a column (or row) divided by the period of the second clock used to perform the transfer (ExtClock). Note that the failure summary correspond to the previous column (or row) tested. The transfer is performed concurrently with the test of a new column or row. The principle of operation is virtually identical to the one explained in LVPAT055 (logic BIST datalogging). See figures 3 and 5 of this application. The serial input can be used to shift in parameter values to be used during the test of the next column or row. There is no data shifted in on figure 5.

One serial input and one serial output is needed. If several controllers are used in parallel, output pins dedicated to each controller are needed. More than one output could be used for each controller. It allows to obtain failure information

## Detailed description of the preferred embodiments

on several memories in parallel or even multiple segments of a same memory. For example, if a memory has 32-bit words and is built as 2 blocks, one containing the first 16 bits of every word and the other containing the last 16 bits of every word, failure summaries can be generated and transmitted on 2 outputs, one for each block. Using multiple outputs for a same memory will maximize the probability of being able to generate a complete bitmap in a single pass at the expense of more silicon area. On the input side, the number of pins depends whether we operate the controllers asynchronously or not. Asynchronous operation involves dedicated serial input for each controller. Synchronous operation involves a pause at the end of a column (or row).

There could be general statistics scanned out at the end of each test phase and/or at the end of the test. For example, the total number of miscompares and/or locations with miscompares. Others are possible. It might use the same pins described above or the normal setup mode.

## Summary contents

The method provides the maximum of information on failures that can be transferred to the tester without having to interrupt the test performed at-speed. In several cases, a complete bitmap indicating the exact location of all failures can be obtained in a single pass. In cases where the density of failures is such that the exact location of each failure can not be transferred to the tester, statistics about the failures and partial information about the location will be transferred instead. This information might still be sufficient in some applications like process monitoring and yield analysis. However, if more information is required, the memory test can be run again to focus on portions of the memory.

A summary is provided for each column during test phases using the column access mode (fast row) and for each row during test phases using the row access mode. The data summary format might be different during the column access mode and the row access mode. This is because there could be significantly less words in a row than there are in a column in embedded memories using very long words. This leaves less time to transfer failure information. Another type of summary could be provided at the end of test phases to report additional information, like the row address of bad rows, location of isolated failures, exact count of various failures instead of ranges, etc... It is also possible to defer unloading this information until the end of the test at the cost of additional on-chip storage.

Several examples of failure summaries are provided in figures 7, 9 and 14. They all use combinations of the following fields i.e. address, failure counts, and failure mask register. Different formats of failure summaries are required to take into account the time available to transfer the summaries during the column access mode and the row access mode. The time varies from one memory to the other and from one mode to the other.

## Detailed description of the preferred embodiments

**Address field :**

-row (or column) address of failures. Since only a limited number of such addresses can be reported, one useful option is to report the first and/or last failure address in the column. This is useful to identify 2 isolated failures and the beginning/end of a group of failures in a column (partial/full column defects). Combined with the information in the failure count fields, a density of failures can be determined. One bit can be appended to each address that an error occurred at the next address to catch 2-cell failures. Only the row (or column) address is necessary since the other components (e.g. test phase and column (or row)) are implicitly known from the time of transfer of the failure summary.

**Failure count fields:**

-Counts for single-bit failures, multi-bit failures ( i.e. 2 or more bit failures but NOT all bits) and massive failures (i.e. at least 1 block of 4 consecutive bits fail, other definitions possible). This will be useful to interpret the test mask register (GOIDs). If there is a high count of single-bit failures and there are more than 1 GOID set. This means that one of the GOIDs correspond to a column failure and the others to isolated failures. On the other hand, a high count of multi-bit failures means that we have 2 or more columns that are bad.

It could be that we don't report the exact value of the count during a column access only at the end. During a column access, we might report exact values up to 3 and then indicate ranges like >25%, >50%, etc... This requires encoding on 3 bits. The encoding could be done for segments (2 or 4 max) of the column because it appears that partial column failure is of interest. Other encodings are possible.

The counts are likely to represent counts of failed locations as opposed to miscompares. It is frequent to have test phases where more than 1 compare occurs per location.

In order to be able to distinguish single-bit from multi-bit failures, a circuit like the one shown in figures 10 to 13. Pipelining is needed for a large number of failure mask register bits (GOIDs). An alternative is to always restrict the number of data output bits processed to 8 (or whatever number seems reasonable). These groups of 8 bits can be tested serially or in parallel. The parallel case requires to generate summaries and to add serial outputs for each group.

The massive failure type can be identified using a number of ways. The goal is to distinguish between failures due to row or column access mechanisms as opposed to individual bits. The criterion needs to take into account partial row or column failures. Such massive failures will tend to affect consecutive bits of a word. For example, a group of 4 adjacent failing bits in the failure mask register could be determined as a massive failure.

## Detailed description of the preferred embodiments

### Failur mask register (GOID fields)

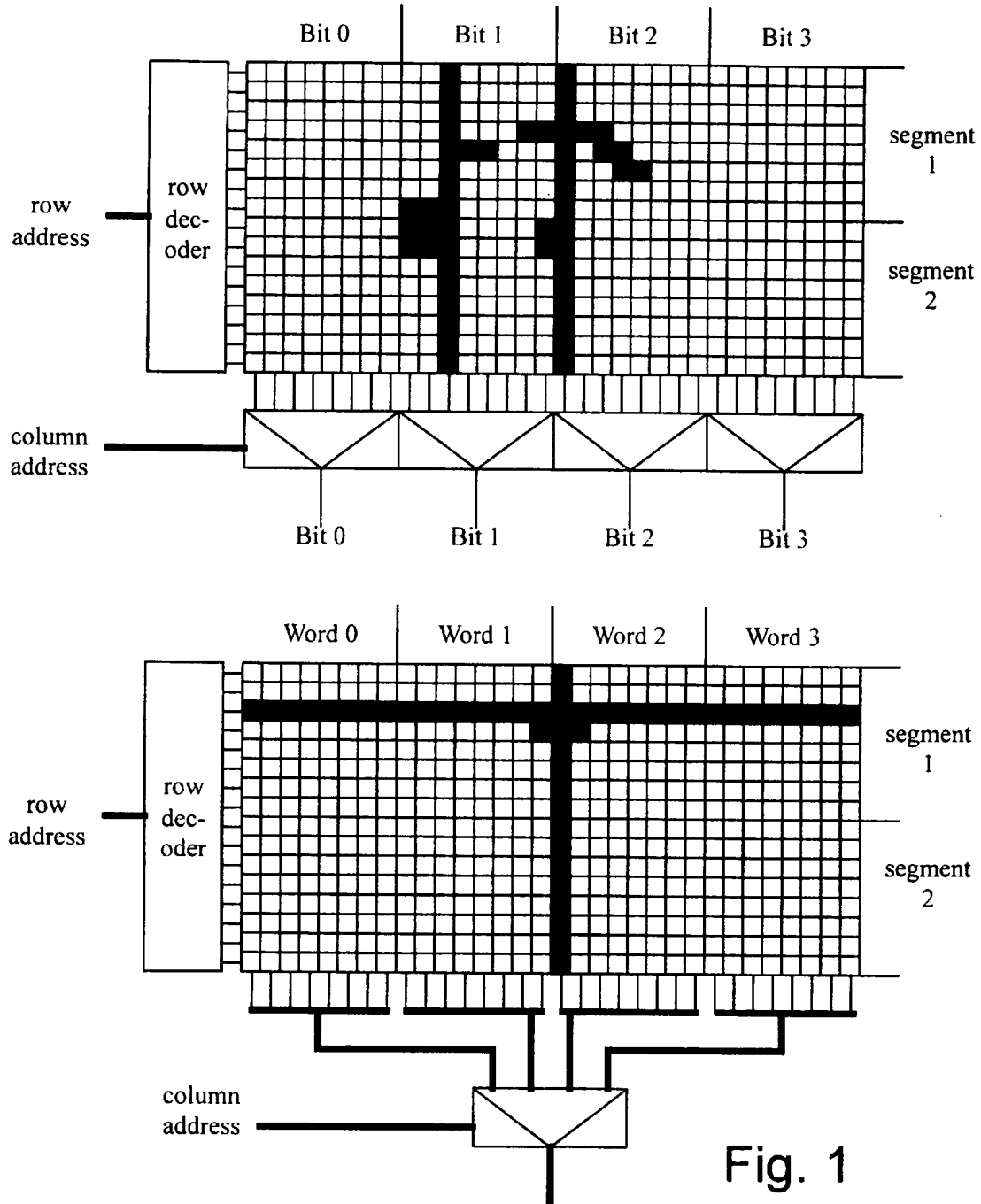
The failure mask register has a memory element associated to one or more comparators, each comparator being associated to one or more memory outputs. The most accurate results are obtained when there is one memory element associate to one comparator and one memory output. A memory element of the register has an active value when a failure occurred on the associated data output(s) since the last time it was initialized. Typically, this register is initialized at the beginning of a new column, in column access mode, or the beginning of new row, in row access mode, but it could be more often if statistics on segments (or portions) of the column (or row) are needed.

For memories with relatively short words (say less than 8 or 16 bits), it is possible to transfer the entire fail mask register. It might even be possible to transfer it more than once for a same column. However, memories with longer words and operating at a relatively high speed with respect to the tester, some trade-offs might be necessary. The possibility of dividing the memory outputs in several groups and generating several failure summaries in parallel has already been mentioned. However, encoding schemes can be used to avoid having to transfer the entire failure mask register. This is because it is expected that relatively few memory data outputs will fail in a same column or row.

For example, say we have 64 GOIDs and that all failures are in the same group of 8 bits, it could be reported in 2 bytes. The first 8 bits indicate the group that have failures and the next 8 bits are the actual GOIDs of that group. If more than one group has failures, there are a few choices. One would be to report only the GOIDs of one of them (always follow the same convention or change the convention based on the test phase to pick up all failures as the algorithm evolves). Other encoding schemes are possible.

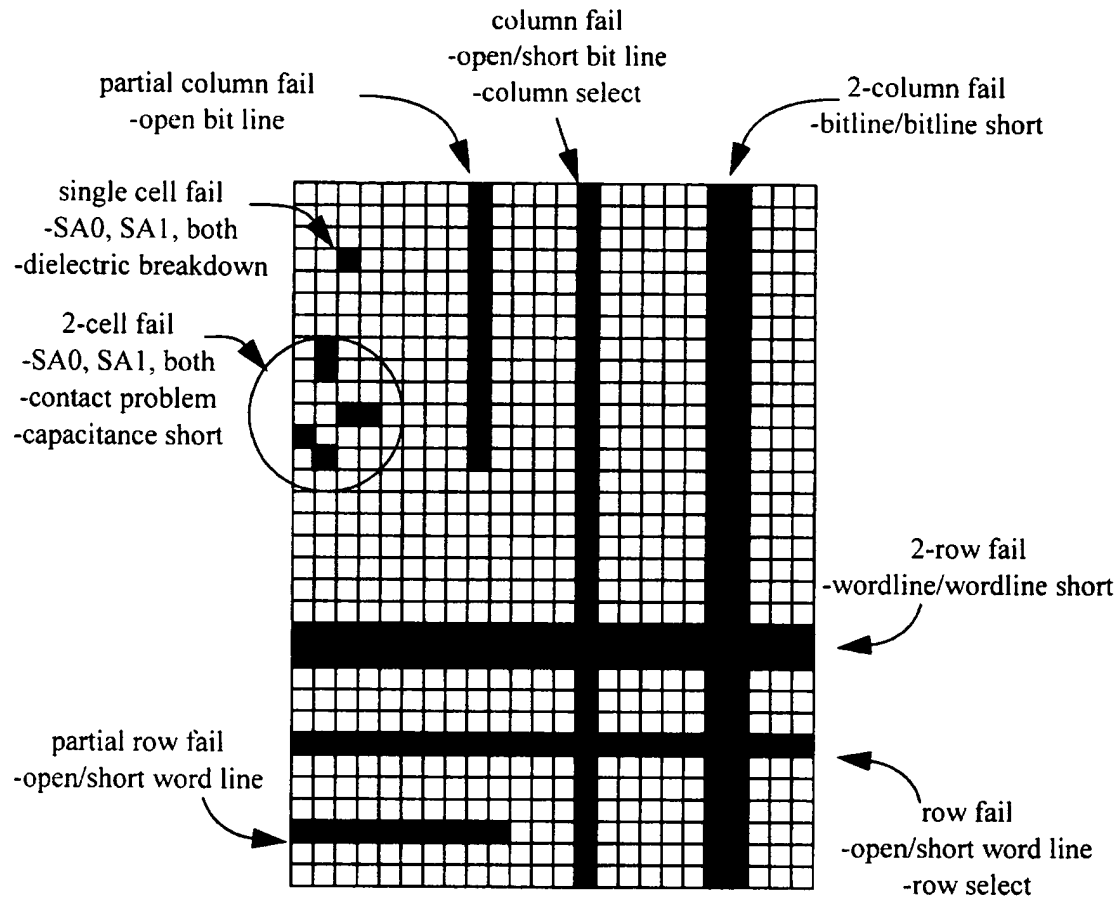
Statistics could be computed per segment on GOIDs. For example, we could count the number of bad GOIDs in a segment. This has the advantage of resolving some ambiguities. For example, suppose we have errors in the first column segment that affect 2 GOIDs and errors in the second segment that affect 2 GOIDs as well but one is common with the first segment. At the end of the column, 3 GOIDs will show errors but the regions of ambiguity are reduced considerably by knowing the number of failing GOIDs on a per segment basis. The count of GOIDs is relatively inexpensive. We can serially count the number of bad GOIDs as we are processing the next segment and/or shifting other fields of the summary, a little bit like in PLLBIST. It involves copying the GOIDs in a shadow register. We can arrange to have less GOIDs than bits in the other field. If there are more GOIDs than bits for the other fields, we have to count by 2 or 4 bits to have the result ready in time.

Detailed description of the preferred embodiments

**Figure 0-1** Memory configurations and failure pattern examples**Fig. 1**

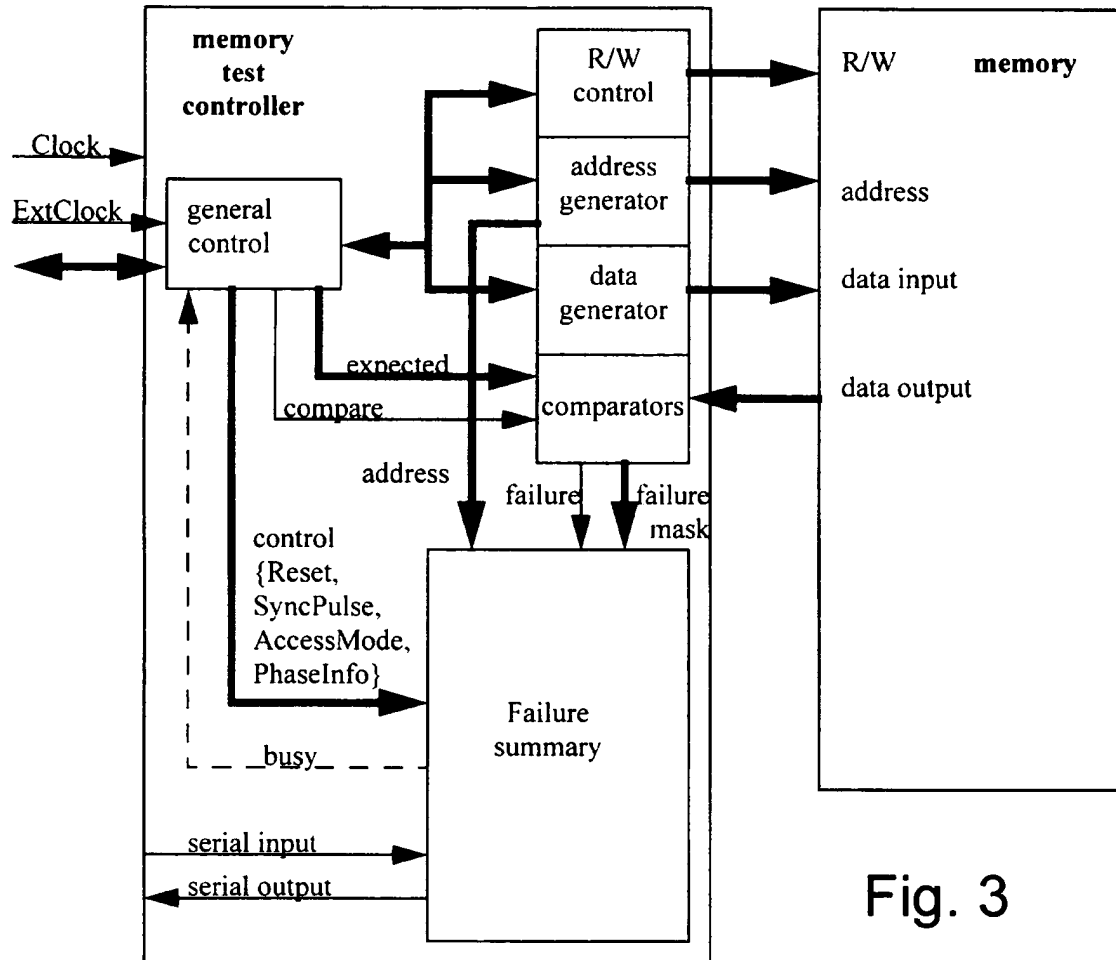
Detailed description of the preferred embodiments

**Figure 0-2** Bitmaps of interest and potential failure causes (SRAM/DRAM)



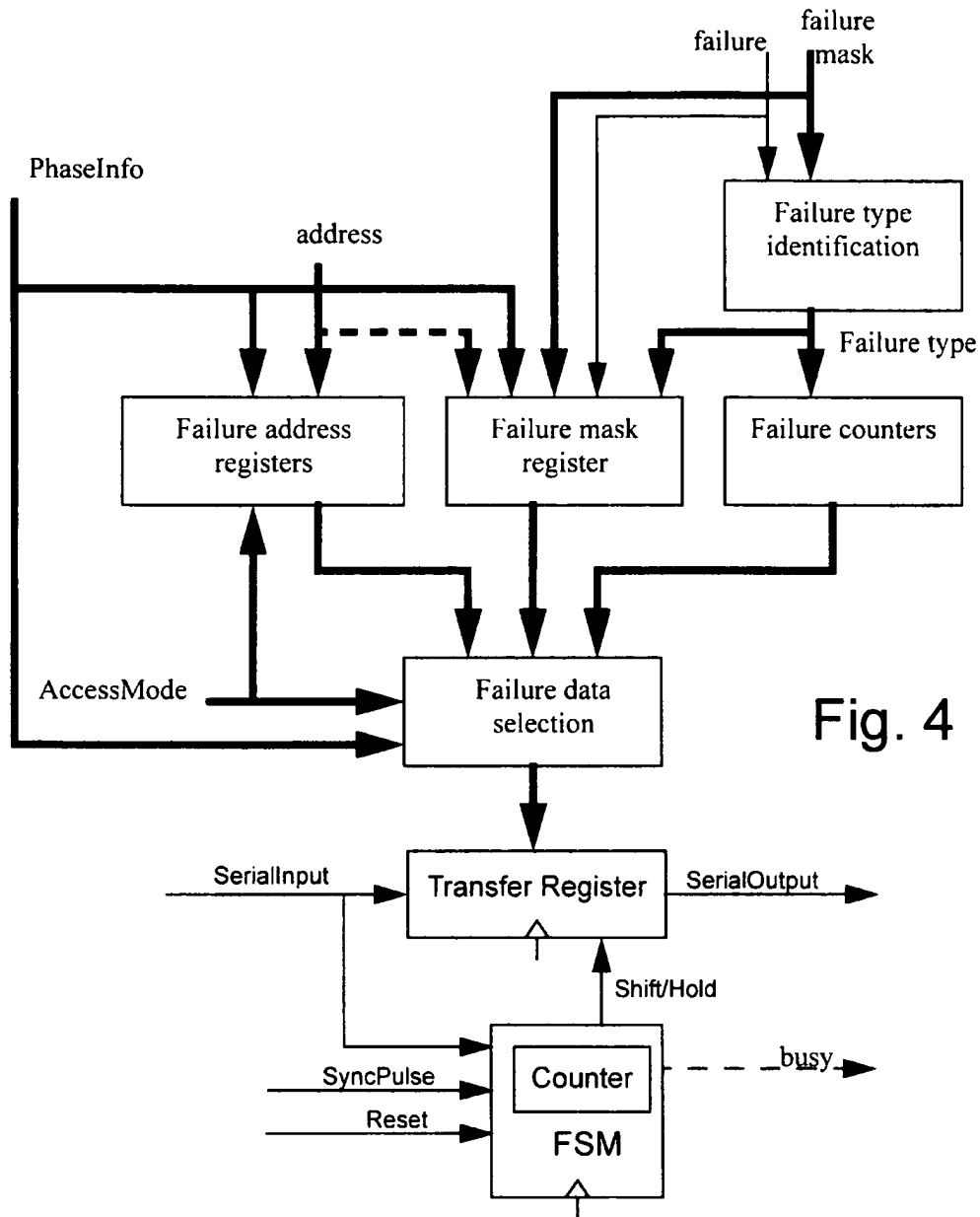
**Fig. 2**

Detailed description of the preferred embodiments

**Figure 0-3** Memory test controller architecture**Fig. 3**

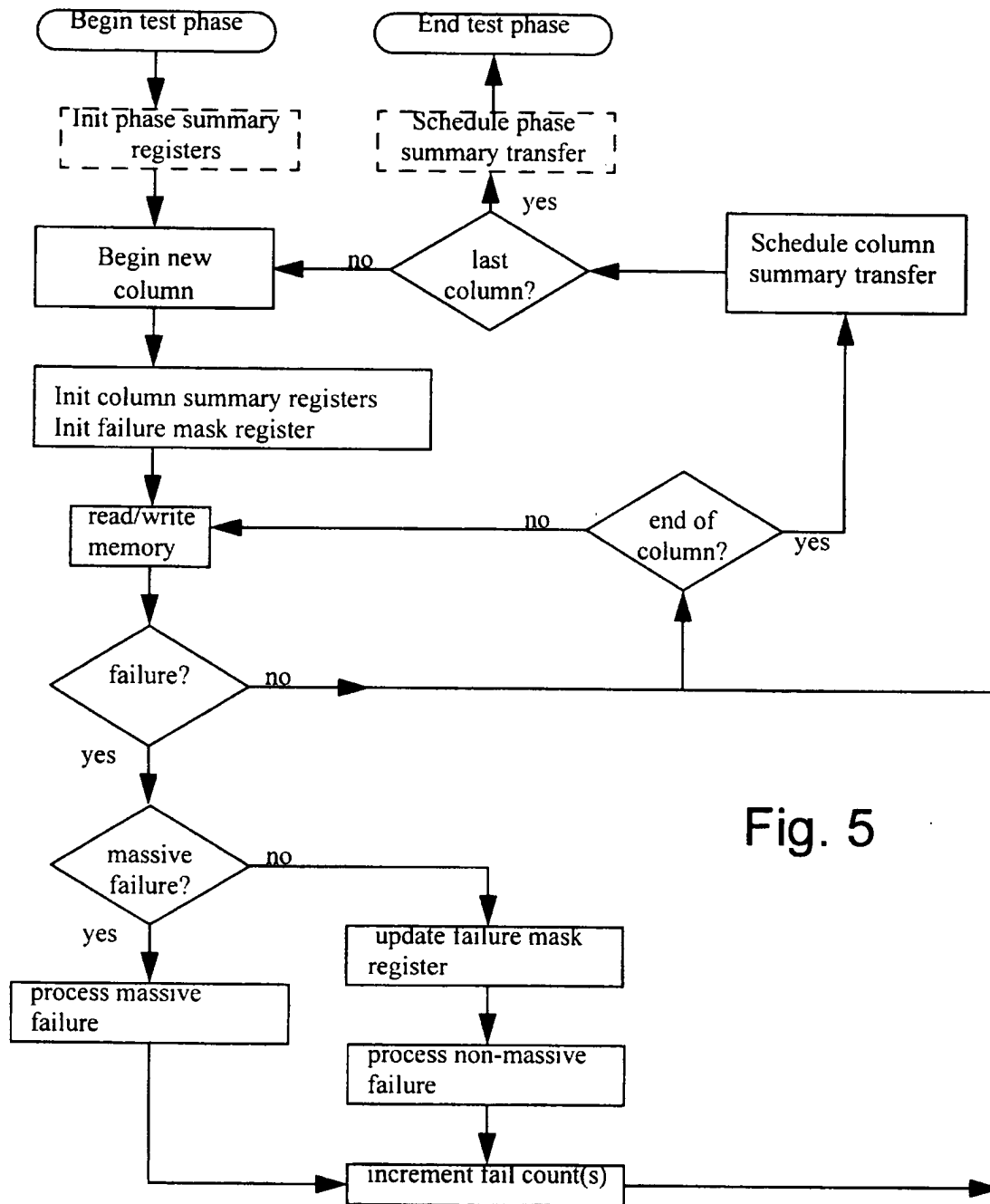
Detailed description of the preferred embodiments

**Figure 0-4** Failure summary generator architecture



**Fig. 4**

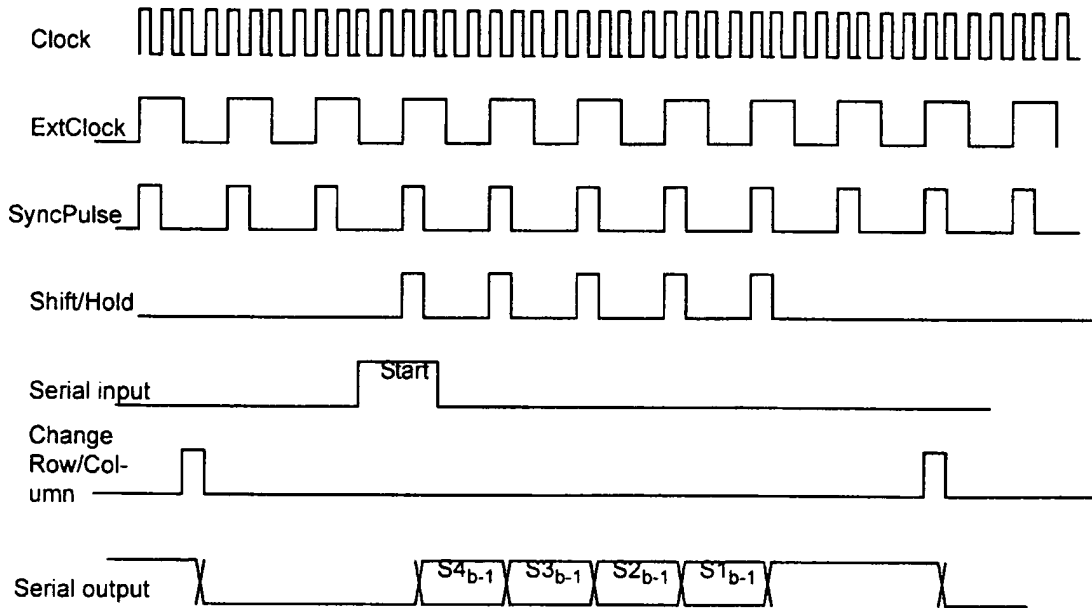
Detailed description of the preferred embodiments

**Figure 0-5** Failure summary generator flow diagram (column and row access modes)**Fig. 5**

Detailed description of the preferred embodiments

**Figure 0-6** Timing of failure summary transfer

---



**Fig. 6**

## Detailed description of the preferred embodiments

**Figur 0-7 Failure summary: Example 1**

Column summary contains row addresses of first and last failure and failure counts

Row summary contains at least failure counts

Phase summary contains failure mask register information

Test is rerun for each failing bit of failure mask register

Column summary

If several banks treated as rows, provide stats for each bank.

Issue, common GOIDs



Row address fields ( $2 \times (8 + 1)$  bits):

Low, high address of failing rows

(excluding massive failure)

flag indicates if address next to those also fail. Can extract 2 double faults.

**Fig. 7**

Failure count fields ( $2 \times 3$  bits)

# of non-massive failures (3 bits)

# of massive failures (3 bits)

Total # of bits: 24 bits

row summary

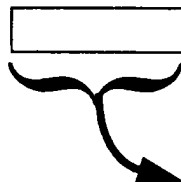
Usually less columns than rows. Could add more stats if time permits



Failure count fields ( $2 \times 3$  bits)

same as above

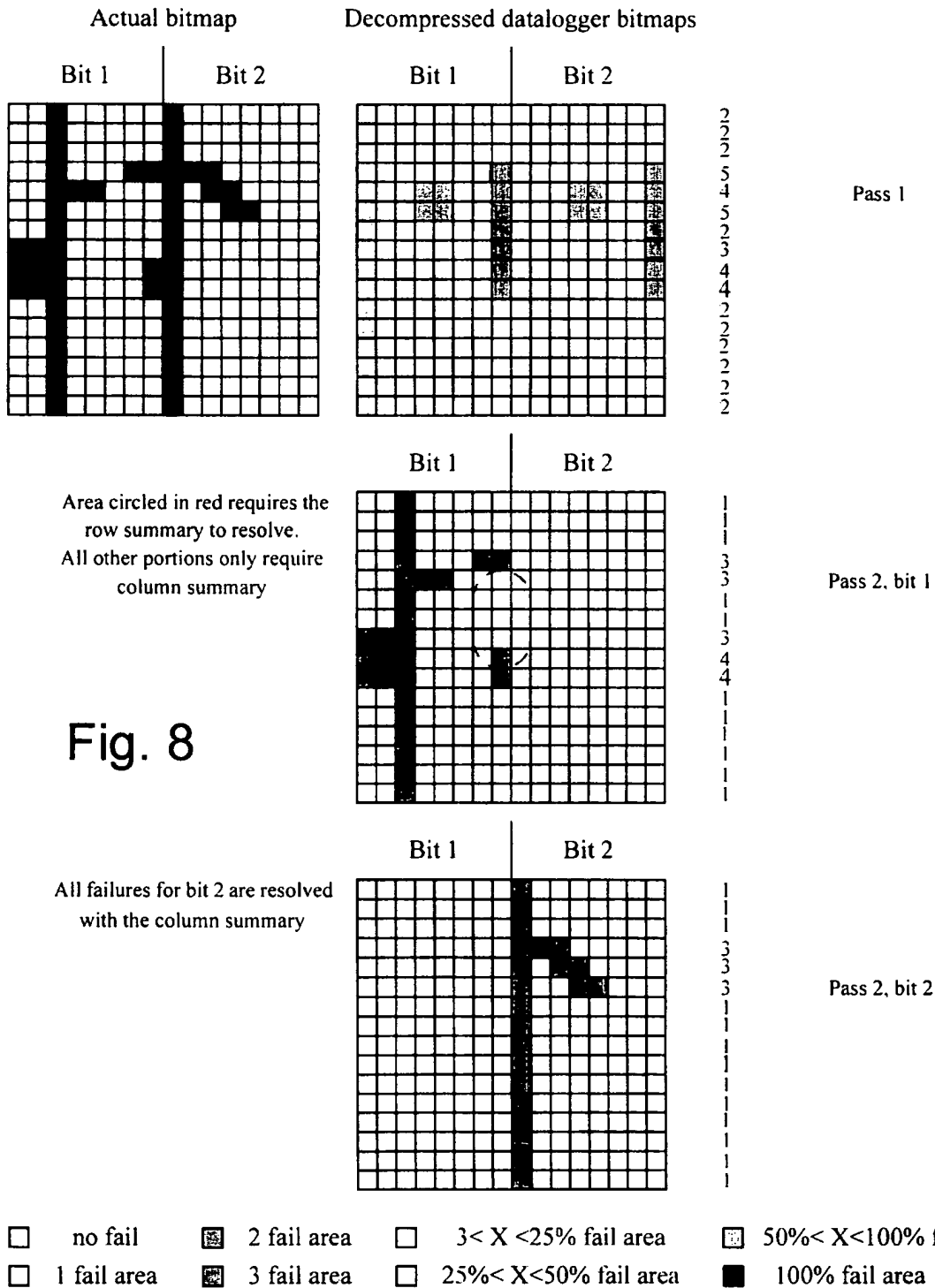
Phase summary (optional)  
GOIDs dumped after each phase



Failure mask register (16 bits)

Detailed description of the preferred embodiments

**Figure 0-8 Results: Example 1: Complex bitmap interpretation (bit grouping = 1)**



**Figure 0-9 Failure summary: Example 2 (no row access mode)**

Column summary contains (for each of 2 address segments):

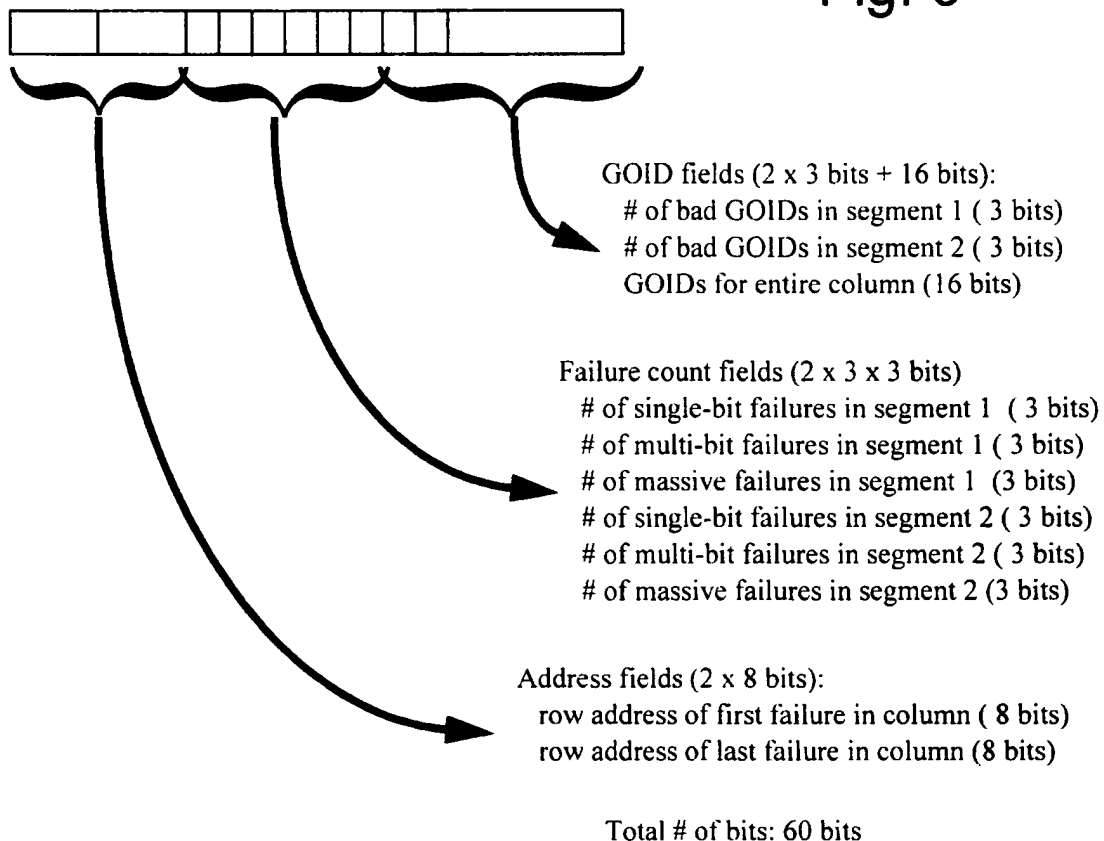
row addresses of first and last failure

failure counts: single-bit, multi-bit and massive failures

failure mask register information: number of bad bits

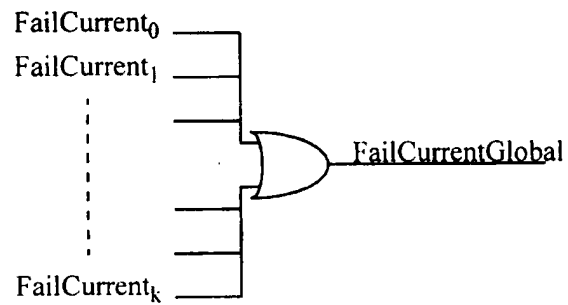
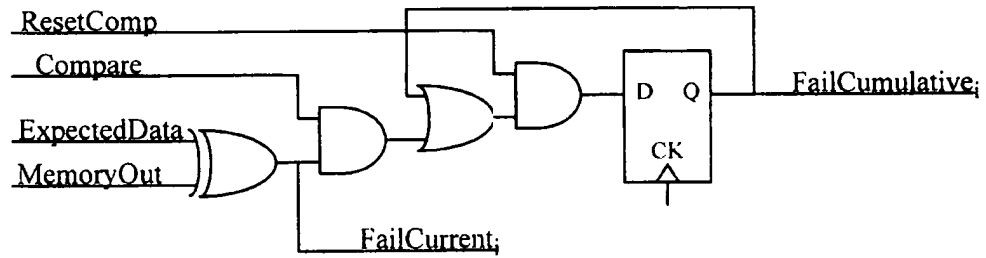
+Failure mask register contents for entire column: encoded for very wide memories. E.g. 64-bit word split into 8 groups of 8 bits. Only transfer register contents for 8 bits and identify group with a 3-bit field.

No row summary

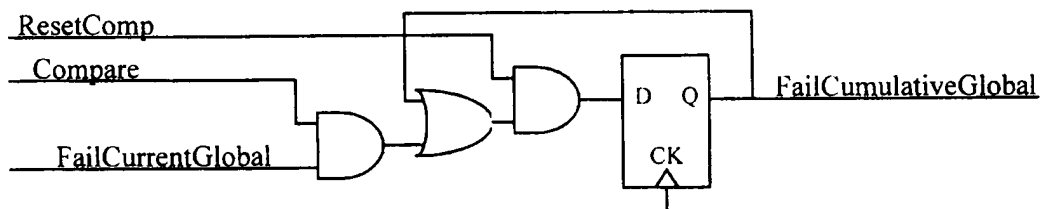
**Fig. 9**

Detailed description of the preferred embodiments

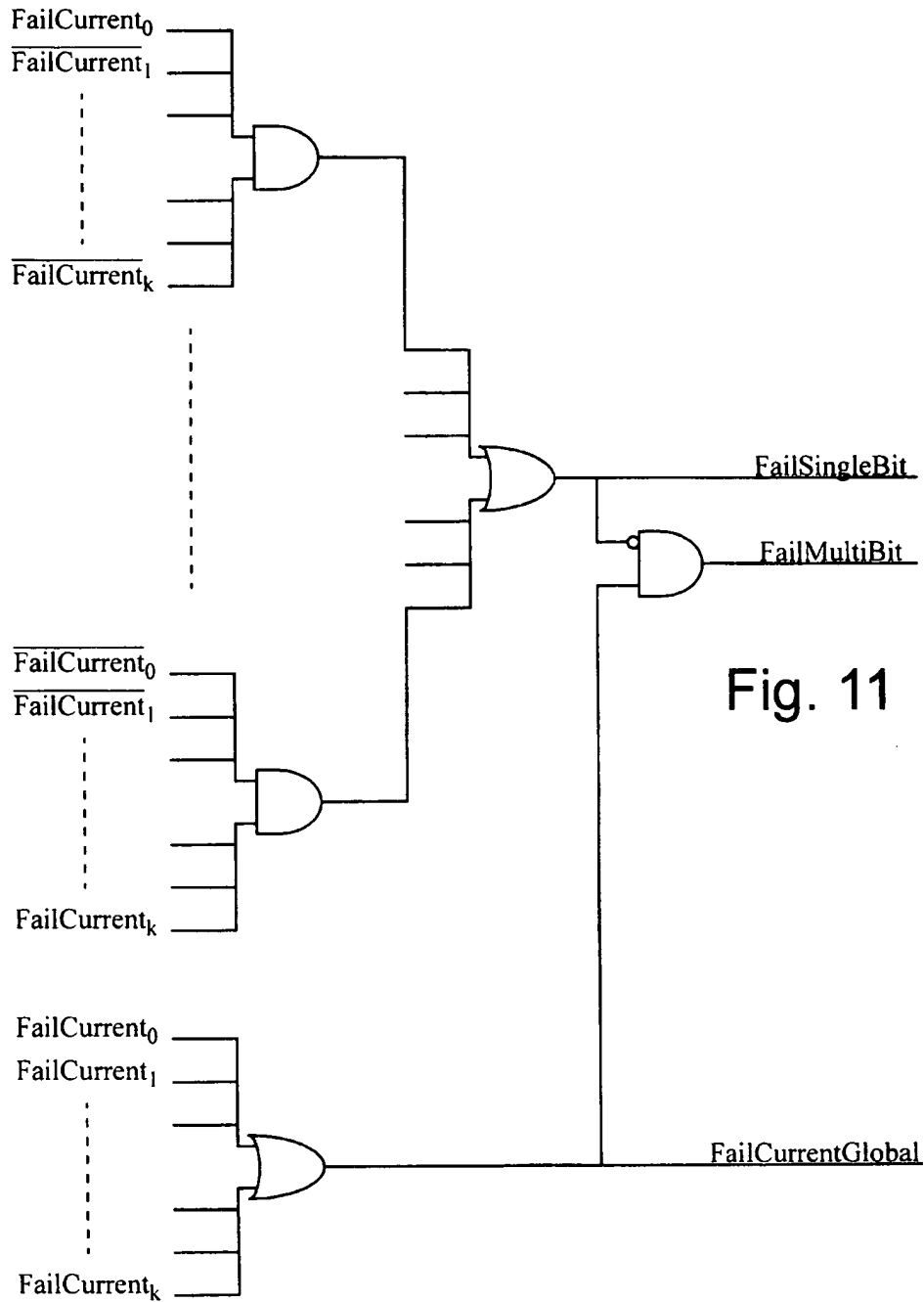
**Figure 0-10** Fail mask and fail mask register



**Fig. 10**

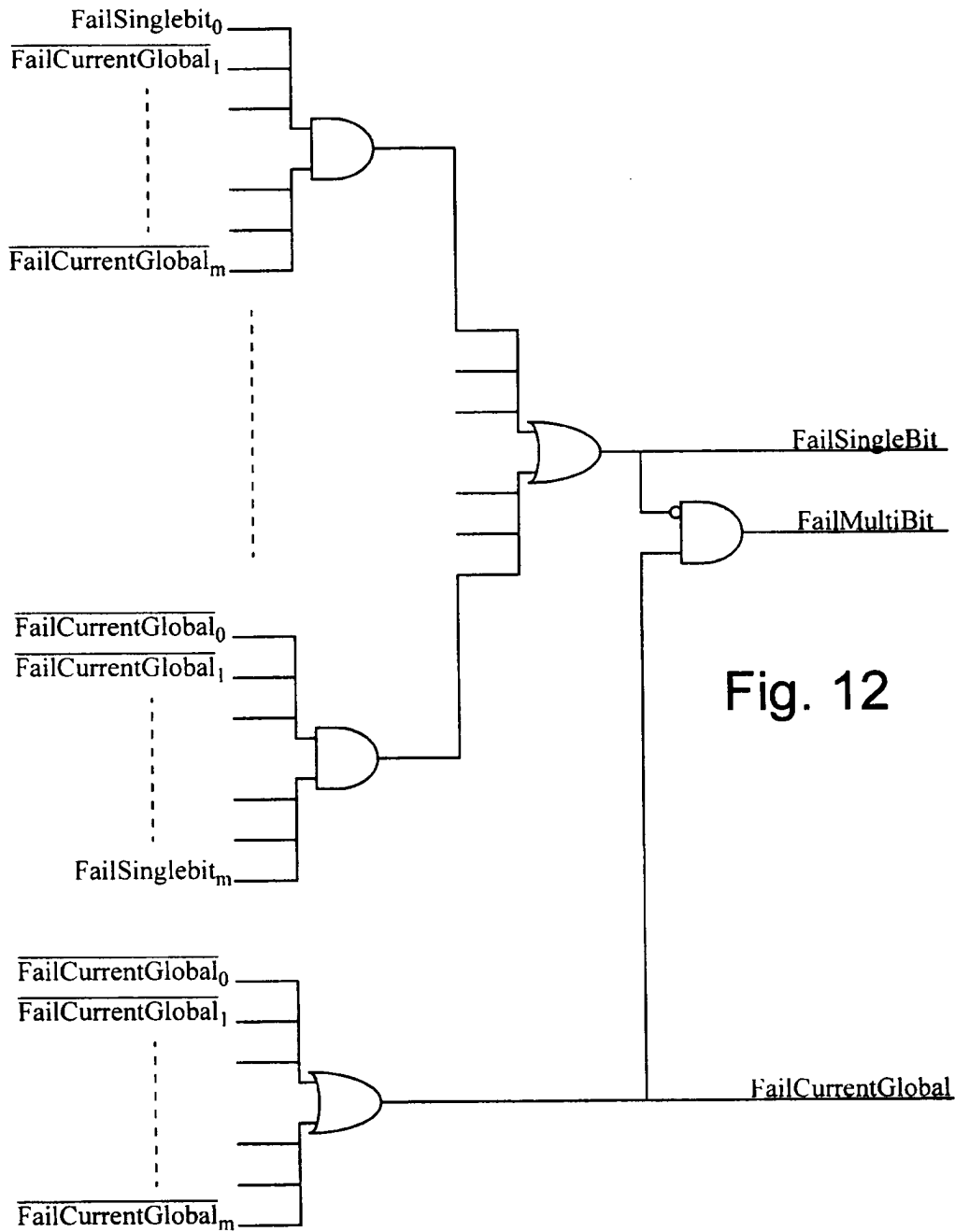


Detailed description of the preferred embodiments

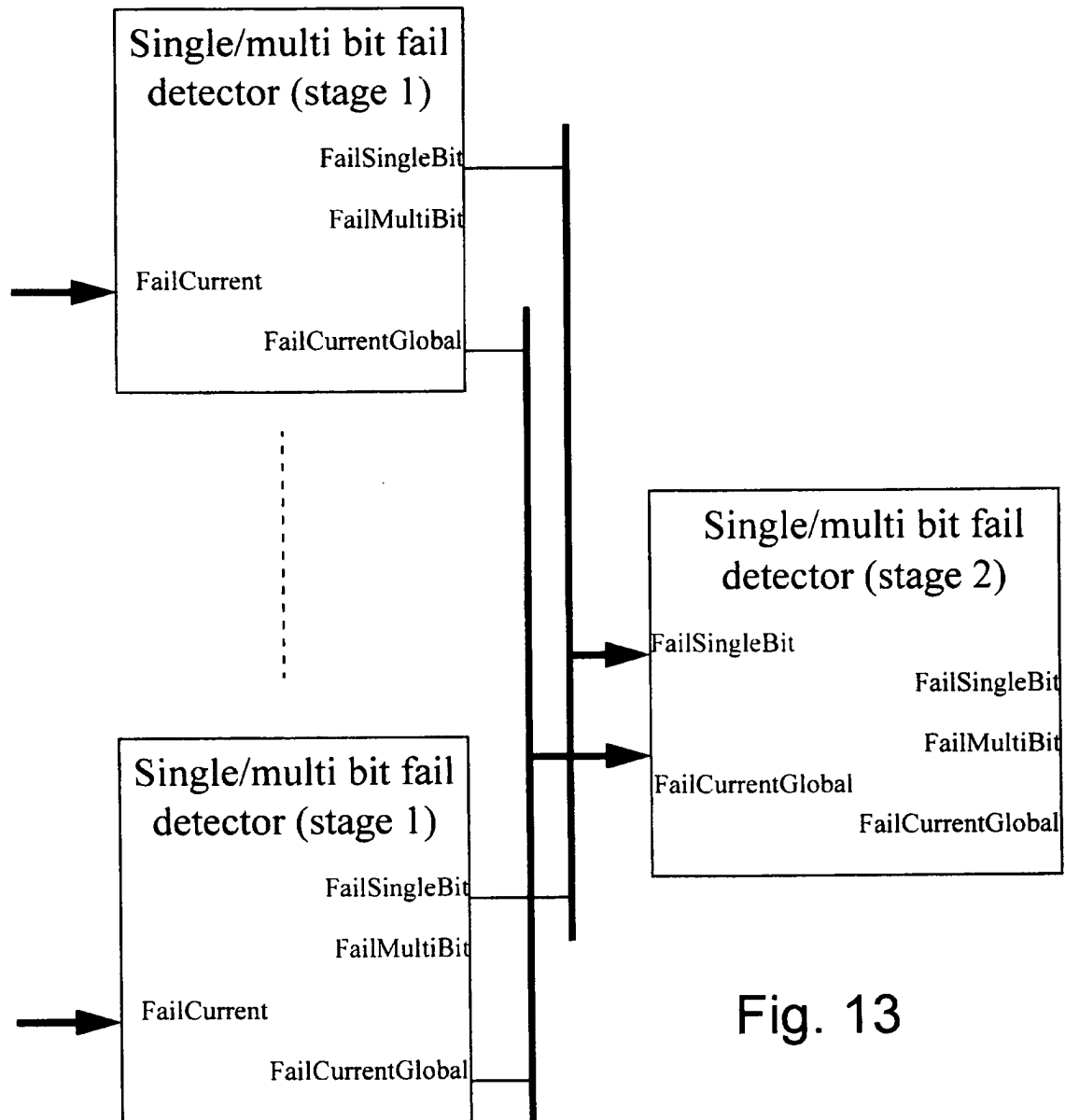
**Figure 0-11** Detection of single-bit and multiple-bit failures (first stage)

Detailed description of the preferred embodiments

**Figure 0-12** Detection of single-bit and multiple-bit failures (second stage)



Detailed description of the preferred embodiments

**Figure 0-13** Cascading single/multiple-bit detectors**Fig. 13**

Detailed description of the preferred embodiments

### Figure 0-14 Failure summary: Example 3

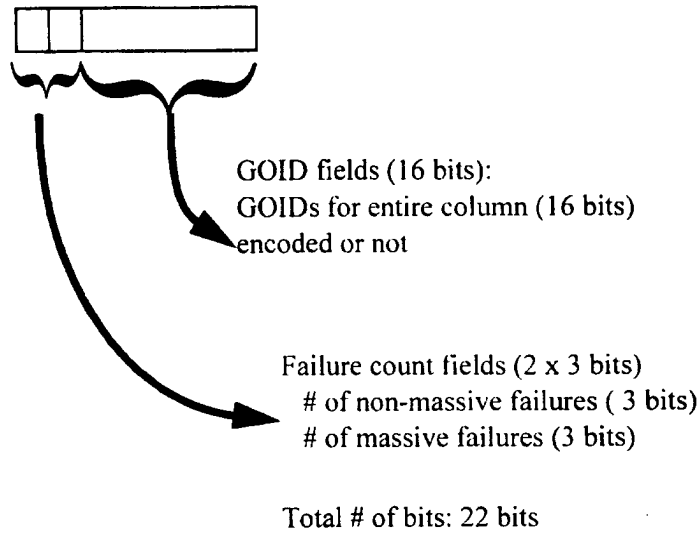
Column summary contains:

failure counts: massive and non-massive

failure mask register information: encoded or not

Row summary contains failure counts only

Column summary



row summary

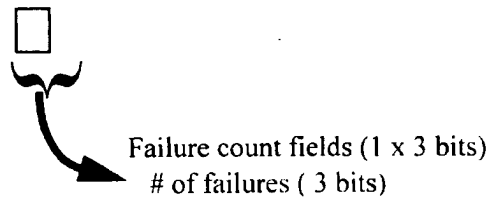


Fig. 14